Advances in Data Mining

Assignment 3 Mastering RandomForest, XGBoost, PCA, LLE and t-SNE

Authors:	*Student number:*
Freek van Geffen	s2633256
Justin Kraaijenbrink	s2577984

Teacher: Dr. W.J. Kowalczyk

December 11, 2020

Introduction

Since the introduction of random forests back in 2001, this machine learning technique for classification and regression tasks has become increasingly popular (Breiman, 2001). Classification tasks are used to predict to which class or category an observation belongs, based on some predictor variables. An example is classification of having a disease or not based on genetic data. Regression tasks have a similar nature in the sense that they are also used for prediction. However, instead of a categorical value we try to predict a value on a continuous scale. For example, think of predicting stock prices based on historical performances. Another technique that is more and more often used, is called boosting. Boosting can also be used for classification and regression tasks. One of the main advantages of both random forests and boosting is the fact that they can quite easily handle high dimensional data. In other words, when there are more predictor variables than observations, random forests and boosting do the job, whereas more classical statistical techniques such as (logistic) regression fail. This is fairly convenient for settings that deal with lots of variables, such as genetic data, image data and health care data.

High dimensional settings are also relatively unsuitable when the goal is not so much prediction but rather visualization. The human brain is able to visualize at most three dimensions, but everything beyond does not work. When we have for example genetic data, it is impossible to visualize that easily without having performed some kind of dimensionality reduction. One of the most well-known techniques for data reduction is Principal Components Analysis (PCA), since 2000 two other popular techniques have been introduced: Locally Linear Embedding and t-distributed Stochastic Neighbor Embedding (t-SNE) (Maaten & Hinton, 2008; Roweis & Saul, 2000). Both LLE and t-SNE are known to be more flexible techniques, often resulting in clearer visualizations. Saillant detail: t-SNE was developed by a Dutchman!

In this report, we will implement all five techniques mentioned above: random forests, boosting, PCA, LLE and t-SNE. For random forests and boosting we employ both regression as well as classification, and for PCA, LLE and t-SNE we only perform classification (regression does not make sense in this case, because we want to see some clustering going on, for which we need categories). In the next sections, we first give a detailed description of the methods used, with details on the data set, study design and some background on the techniques themselves. Thereupon we provide an extensive discussion of the results we obtained, to end this report with some concluding remarks.

Methodology

Data

For this research we made use of two separate data sets: one for regression and one for classification. For the regression tasks, we used data on housing prices retrieved from `https://www.kaggle.com/ashydv/housing-price-prediction-linear-regression/notebook?select=Housing.csv`. We used `price` as outcome variable and all 12 other variables, such as `area`, `number of bathrooms` and `airconditioning`, as predictor variables. There were 545 observations and no missing values. The classification tasks were performed using a dataset on different types of breast cancer, retrieved from `https://www.kaggle.com/piotrgrabo/breastcancerproteomes`. The dataset consisted of 12,550 protein intensities (predictors) for 105 observations, but there were parts of the data missing. Since the main goal of this research was to implement several machine learning techniques, we decided to only use complete data. The final dataset which we used for classification contained data on 7,994 protein intensities for 80 observations. For the outcome variable there were four types of breast cancer: Basal-like, HER2-enriched, Luminal A and Luminal B.

Study Design

In this research, the main interest was in implementing random forests, boosting, PCA, LLE and t-SNE. To do so, we performed the following steps:

1. **Regression analysis**
   (a) We first performed an exploratory data analysis to see whether we should do some more advanced data cleaning. A linear model was fit to the data with house price as outcome variable and all other twelve variables as predictor variables. Note that there were some categorical variables in our data set, so we converted them to dummy variables (one-hot encoding). To see whether there were any abnormalities, we made a histogram of the residuals as well as pairwise scatterplots of the continuous variables.

   (b) In order to be able to compare the different techniques, we split the data into a train set (80%) and test set (20 %). The train set was used to fit the models, the test set was used to assess the predictive performance of the different techniques. As measure of predictive performance we used the proportion of explained variance, defined as

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)}{\sum_i (y_i - \bar{y}_i)}$$

   (c) The next step was fitting a baseline model, to which we could compare the results of random forests and boosting. As baseline model we used multiple linear regression, where we first fitted a model with all predictor variables. Subsequently we did some kind of model selection by removing all non-significant predictors. Performance of this baseline model was assessed on the test data.

(d) The fourth step in the regression analysis was to perform random forests. Although random forests is known to perform fairly well with the default settings, the best results are obtained by carefully tuning the hyperparameters. We therefore exploited a two-step hyperparameter tuning approach using random grid search. The first round inspected a particularly wide and rough parameter space, whereas the second round was used to narrow down the parameter space to obtain more precise estimates. With the optimal parameters the final random forest model was fitted and used for prediction on the test set.

(e) The final step was to perform boosting. The performance of boosting algorithms is highly related to having the correct hyperparameters. Thus, parameter tuning is important step and a large amount of important variables can be tuned. Tuning all these parameters at once leads to enormous computing time. Therefore, a 5-step sequential grid search approach is executed to find the optimal parameters. The optimal parameters are used to fit a final model, which is used to predict on the test set.

2. **Classification analysis**

   (a) For the classification analysis, we used a similar approach, so we first performed exploratory data analysis, divided the data into a train set and a test set, and fitted a baseline model. Exploratory data analysis for high dimensional data is a bit different than for low dimensional data, so we made histograms of average protein intensities and participant averages to check for outlying observations. The baseline model we used was multinomial logistic lasso regression. As measure of performance we utilized plain accuracy, defined as the percentage correctly classified observations.

   (b) Random forests and boosting were exploited in the same way as for regression analysis. As measure of predictive performance we used the accuracy.

   (c) Furthermore, PCA, LLE and t-SNE were implemented to see the difference in data visualizations between these techniques. PCA does not depend on any hyperparameter, but LLE and t-SNE do so. The most important parameter to tune is the number of neighbors, also called perplexity, which we investigated using visualizations for different values of this perplexity.

Techniques

In this section we provide some background information about each of the five techniques and dive more into the inner workings of the algorithms.

Random forests

A true forest consists of a huge collection of several trees, and this is exactly how we can see random forests as well. Random forests are based on the concept of decision trees, which partition the data into non-overlapping areas. Let's illustrate this with a simple example, which is graphically depicted in Figure 1. Say we want to predict house prices based on the total area, where we expect larger houses to have higher prices. A decision tree can create a

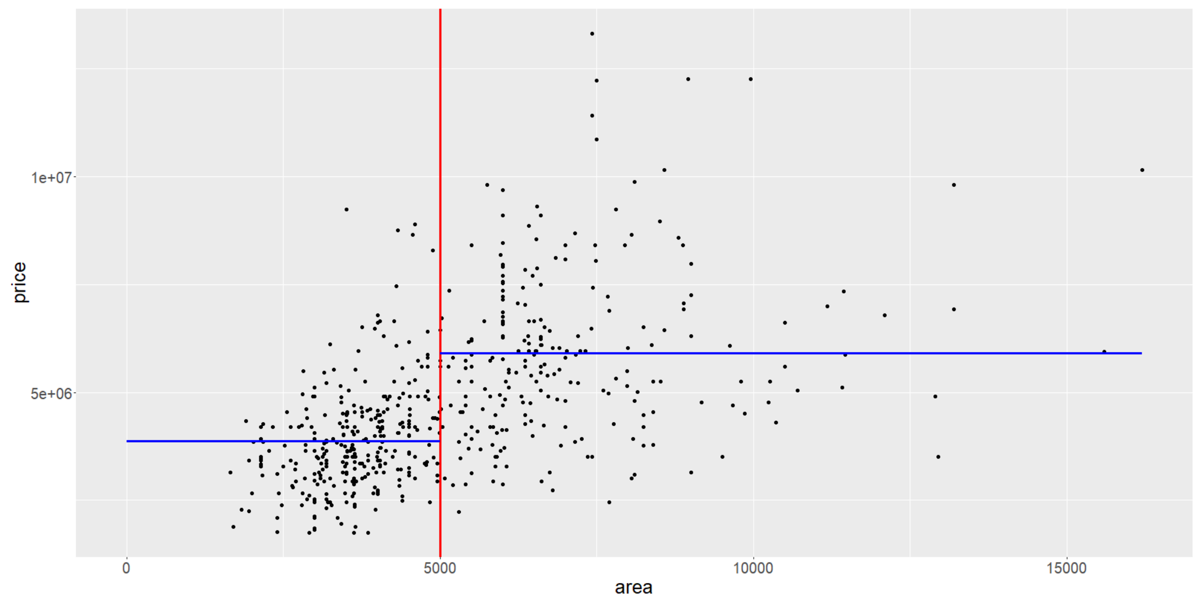


Figure 1: Illustration of the decision tree rationale. Image

split point for the predictor `area` (red line in Figure 1). All observations that have an area that is smaller than that split point belong to the part on the left, whereas all observations with a larger area belong to the right part. The average house price of all observations within each part is presented by the horizontal blue lines and can be used as prediction value for new observations. This idea can be easily extended to multiple split points and multiple predictor variables. The splits are determined by minimizing a loss function, also known as impurity measures. Commonly used measures are:

$$MSE = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{y}_m)^2 \quad \text{(regression)}$$

$$Gini = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \quad \text{(classification)}$$

where N_m the number of final areas into which the data is partitioned, R_m particular area m, y_i the actual observation value, $\hat{y}_m$ the predicted value, K the number of categories and $\hat{p}_{mk}$ the proportion of category k observations in area m.

Now, the idea of random forests is to fit a whole set of decision trees and combine the results. This works as follows:

We implemented random forest using `RandomForestRegressor` and `RandomForestClassifier`. Important hyperparameters to tune in random forests are the number of trees you fit (`n_estimators`), the depth of the tree (`max_depth`), the minimum number of samples for a split (`min_samples_split` and the minimum number of samples in the final nodes of the tree (`min_samples_leaf`). We made use of `RandomizedSearchCV` to find optimal values for these hyperparameters.

Algorithm 1 Random forests

```
1: Draw a bootstrap sample of size $N$ from the train data
2: Randomly select $m$ from all $p$ predictor variables
3: From the set of selected variables, pick the variable for which the loss-function decreases most
4: Use this variable to determine a splitting point
5: Repeat steps 1 - 4
6: Aggregate all the bootstrap trees by takin the mean (regression) or majority class (classification)
```

XGBoost

The XGBoost algorithm is a specific type of boosting algorithm. Boosting is a supervised learning method which can be used for classification and regression tasks. The main idea behind boosting is that there is a sequence of classifiers $(f_0, ..., f_M)$ which are trained sequentially and each classifier focuses (f_i) on the errors made by the previous classifier (f_{i-1}). This ensemble of sequential fitted weak learners is able to reduce bias and variance, which results in a classifier with a very high prediction accuracy. The XGBoost, Extreme Gradienst Boosting, algorithm focuses on the errors made by the previous classifier by adjusting the response variable. Algorithm 2 shows the steps in the XGBoost algorithm. In this report the XGBoost algorithm is implemented using the `xgboost` library in Python.

Algorithm 2 XGBoost algorithm

```
1: Initialize $f_0(x) = \text{argmin}_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2: for $m = 1, ..., M$ do
3:     for $i = 1, ..., N$ do
4:         $r_{im} = -\alpha[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}]_{f=f_{m-1}}$
5:     end for
6:     Fit a regression tree to targets $r_{im}$ giving terminal regions $R_{jm}, j = 1, ..., J_m$.
7:     for $j = 1, ..., J_m$ do
8:         $\gamma_{jm} = \text{argmin}_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$.
9:     end for
10:    Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
11: end for
12: Output $\hat{f}(x) = f_M(x)$.
```

Tuning all XGBoost parameters all at once results in a grid search with overwhelming computing time. Therefore, we decided to use an sequential approach such that we are able to tune all the important parameters but without having an enormous computing time. Table 1 shows which parameters are tuned, their function, their tuning range, and in which grid they will be tuned.

Parameter	Function	Range	Grid
`Max_depth`	Maximum depth of tree	$1, 2, ..., 10$	1
`Min_child_weight`	Minimum sum of weights in a child	$0, 1, ..., 10$	1
`Min_split_loss`	Minimum loss reduction for a split	$0.1, 0.02, ..., 2$	2
`Subsample`	ratio of train observation used to create trees	$0.05, 0.1, ..., 1$	3
`Colsample_bytree`	ratio of columns used to create trees	$0, 0.05, ..., 1$	3
`reg_alpha`	L1 regularization term	$0, 10^{-5}, 10^{-4}, ..., 10^{3}$	4
`Learning_rate` (α)	How much the weights are adjusted each time	$0.01, 0.02, ..., 0.2$	5
`n_estimators`	The number of trees that will be created	$50, 100, ..., 1000$	5

Table 1: Overview of the tuned parameters with function, tuning range, and in which grid they are optimized.

First, we run the model with defaults settings and `n_estimators` = 10.000 using the `XGBoost.cv` function to get an initial amount of estimators which will be used for further grid search. Subsequently, each grid search with 5 fold cross-validation is executed sequentially using `GridsearchCV`, and after each grid the optimal amount of estimators is re-calibrated while using the optimal parameters from the last grid search. Re-calibrating the number of estimators after each grid search makes the grid searches more efficient. Subsequently the final XGBoost model is fitted using the optimal parameters from grid 5.

For the regression task, the objective of the XGBoost algorithm is minimizing the squared error, and GridsearchCV function maximizes the explained variance. For the classification task, the objective of the XGBoost algorithm is minimizing the softmax and the objective of the GridSearchCV function is maximizing the accuracy.

PCA

Principal Component Analysis is a well-established technique used for dimensionality reduction. It projects points in the p-dimensional space to a lower m-dimensional space. Without going into too much detail, the method tries to find new variables, such that the first variable maximizes the variances of the data and the second variable has the second most variance, et cetera. PCA was implemented by means of `PCA`.

LLE

One of the disadvantages of PCA is that it is a linear method and therefore not suitable for mapping nonlinear manifolds to lower dimensional space. A dimensionality reduction method that can deal with nonlinearity is locally linear embedding. LLE is based on the idea that each data point and its closest neighbours lie on a locally linear segment. Now, each data point can be reconstructed from the point in its neighborhood, and we can optimize the cost function

$$\mathcal{E}(W) = \sum_i |\mathbf{x}_i - \sum_j W_{ij}\mathbf{x}_j|^2$$

. Here, weights W_{ij} represent how much neighbor j adds to the reconstructed point i (Saul & Roweis, 2000). This problem is undefined because there are more parameters to estimate than function to solve, so we add a pair of constraints: 1. for the reconstruction of each data point

$\mathbf{x}_i$, we only use its k neighbors, where k is user-specified, and 2. the sum of all weights equals 1.

The concluding step of the algorithm is mapping the high dimensional vector $\mathbf{x}_i$ to a lower dimensional vector $\mathbf{y}_i$, by minimizing

$$\Phi(y) = \sum_i |\mathbf{y}_i - \sum_j W_{ij}\mathbf{y}_j|^2$$

Here, the weights remain fixed so we can optimize $\mathbf{y}_j$

The LLE algorithm was implemented using `LocallyLinearEmbedding`.

t-SNE

This relative new dimension reduction method visualizes high-dimensional data by giving each datapoint a location in a two or three-dimensional map. High-dimensional Euclidean distances between datapoints are transformed into conditional probabilities that represents similarities. T-SNE uses the student t-distribution with a symmetric version of the cost function. The conditional probability $p_{j|i}$ and lower dimensional similar conditional probability $q_{j|i}$ are given by

$$p_{j|i} = \frac{\exp\left(-||x_i - x_j||^2/2\sigma_i^2\right)}{\sum_k exp(-||x_i - x_j||^2/2\sigma_i^2)}$$
$$q_{j|i} = \frac{exp(-||y_i - y_j||^2)^{-1}}{\sum_k \exp\left(-||y_i - y_j||^2\right)^{-1}}$$

The t-SNE algorithm minimizes a single Kullback-Leibler divergence between a joint probability distribution, P, in the high-dimensional space and a joint probability distribution, Q, in the low-dimension space:

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}}$$

The objective function is minimized using a gradient descent method. Subsequently, a binary search for the value of σ_i is executed. In order to find σ_i, the perplexity should be specified. The perplexity P can be seen as the number of neighbors that are relevant to compute the conditional probabilities and a higher perplexity results in a higher variance. The t-SNE solution is optimized by adding an exponentially decaying sum of previous gradients, which results in the following update rule:

$$Y^t = Y^{t-1} + \eta \frac{\partial C}{\partial Y} + \alpha(t)(Y^{t-1} - Y^{t-2})$$

Where Y^t represents the solution at iteration t, η the learning rate, and $\alpha(t)$ the momentum at iteration t.

The parameters η and P will be tuned. The tuning process can not be automated and the optimal parameter settings will be selected based on class separability in the representation of the solutions. The t-SNE algorithm is implemented using `Manifold` library from Sklearn and is executed on the cancer disease dataset.

Results

In this section we first present the results of random forests and boosting for the housing data (regression), to follow up with random forests, boosting, PCA, LLE and t-SNE for the breast cancer data (classification). For both regression and classification tasks we compare the results with a baseline model.

Regression

Baseline: exploratory analysis and multiple linear regression

We first conducted some exploratory data analysis by inspecting pairwise scatterplots for all continuous variables in our data set, where we did not observe any abnormalities. An example of such a pairplot is presented in Figure 2, from which we can also already observe that the outcome variable `price` is a bit skewed to the right, and there is a positive correlation between `area` and `price`.

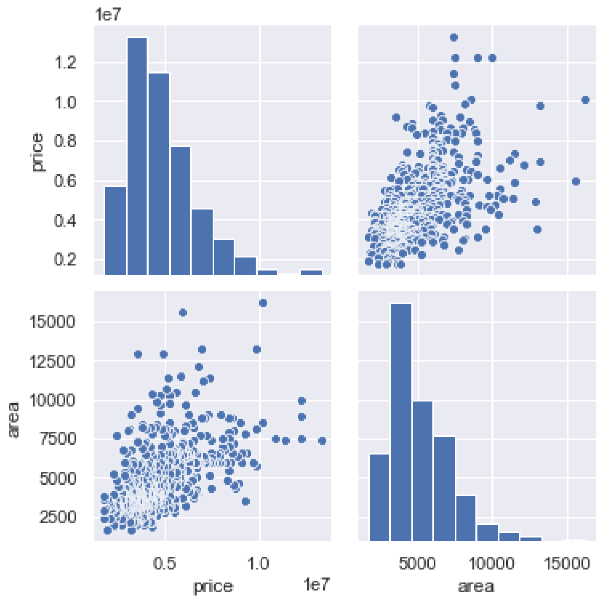


Figure 2: Pairwise scatterplots for `price` and `area`

We also investigated the residuals to see if there are any violations of the assumption that all errors are normally distributed around zero for linear regression. From Figure 3 we conclude that there are no severe deviations from normality. We see one observation that could be a potential outlier, because it lies more than four standard deviations away from the mean, but since we don't know the underlying mechanism we decided to run the analysis with all observations included.

After the exploratory data analysis we turned to fitting a linear model, with all predictor variables included. We decided to narrow down our model by selecting only the relevant (i.e. significant) predictors, in order to make the model more generalizable. Results of the final model are presented in Table 2. Prediction on the test set yielded an explained variance of .684, which is actually quite good!

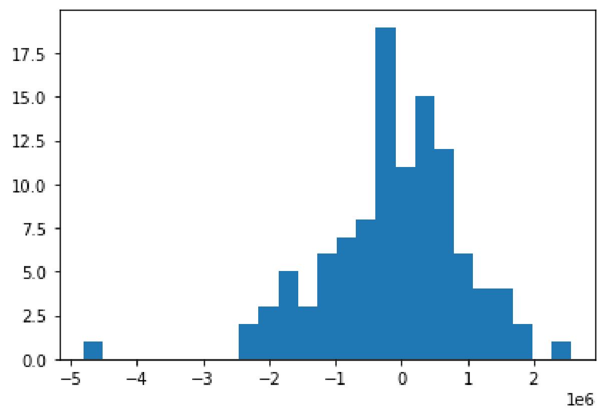


Figure 3: Histogram of residuals to inspect assumptions for the linear model.

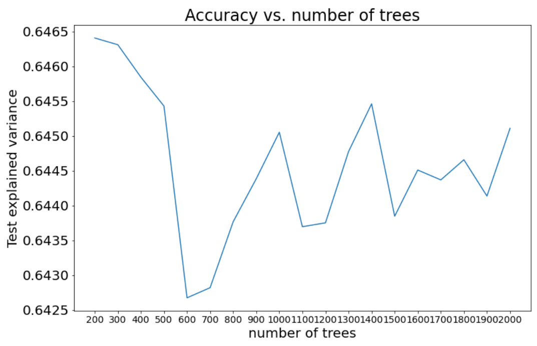
Variable	β	SE	Std. β	p
intercept	-397.03	2.15e5		.999
area	232.23	27.80	.26	.000
bathrooms	9.98e5	1.13e5	.27	.000
stories	4.48e5	6.72e4	.21	.000
parking	2.92e5	6.59e5	.13	.000
mainroad_yes	3.85e5	1.60e5	.07	.017
guestroom_yes	3.39e5	1.47e5	.07	.022
basement_yes	3.34e5	1.26e5	.08	.008
hotwaterheating_yes	8.69e5	2.47e5	.10	.000
airconditioning_yes	9.30e5	1.23e5	.23	.000
prefarea_yes	7.31e5	1.31e5	.16	.000
furnishingstatus_furnished	4.12e5	1.42e5	.10	.004
furnishingstatus_semi-furnished	3.17e5	1.24e5	.08	.011

Table 2: Multiple linear regression results

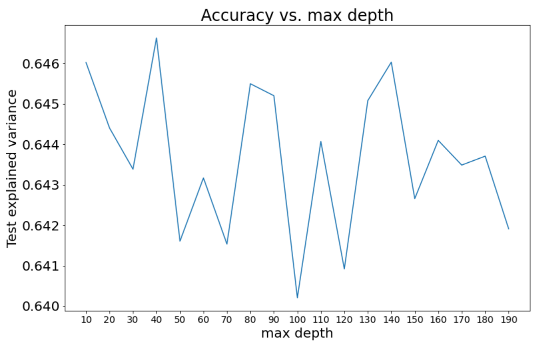
Random forests

In order to implement the random forests model, we first conducted two rounds of hyper parameter tuning using random grid search with 5-fold cross-validation and 100 iterations. The optimal parameters we found were `n_estimators` = 200, `max_depth` = 140, `min_samples_leaf` = 3 and `min_samples_split` = 7. Figure 4 shows how the the explained variance changes in the parameter space for each hyper parameter.

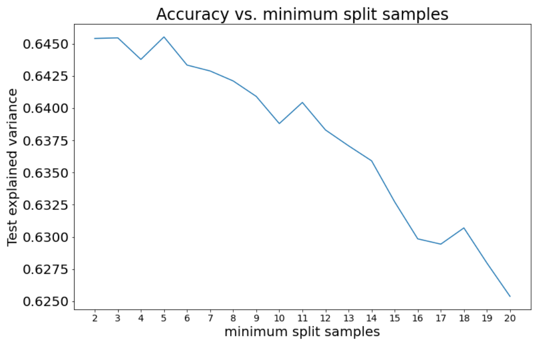
We see that the performance decreases when the number of samples per split increases, and the same holds for the minimum samples per leaf. This makes perfect sense, since the larger the number of samples per leaf, the more global the prediction becomes. We also see that the number of trees does not influence the solution much, which is also true for the maximum depth. This can be explained by the fact that the number of predictors is relatively small, which means that there are no super complex trees required. The final model yielded an explained variance of .623 on the test set.



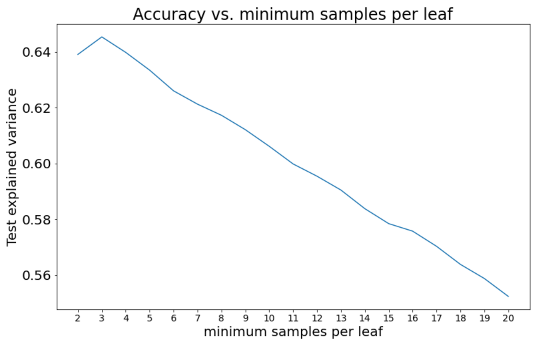
(a) `n_estimators`



(b) `max_depth`



(c) `min_samples_split`



(d) `min_samples_leaf`

Figure 4: Explained variance as function of a. the number of trees, b. the maximum depth, c. the minimum number of samples per split and d. the minimum number of samples per leaf.

One way of interpreting the results of random forests is to look at relative variable importance. These are presented in Figure 5 and it seems that especially `area` and the number of bathrooms are important predictors. These predictors were also found by linear regression as most important ones, which can be retrieved from their largest standardized coefficients presented in Table 2.

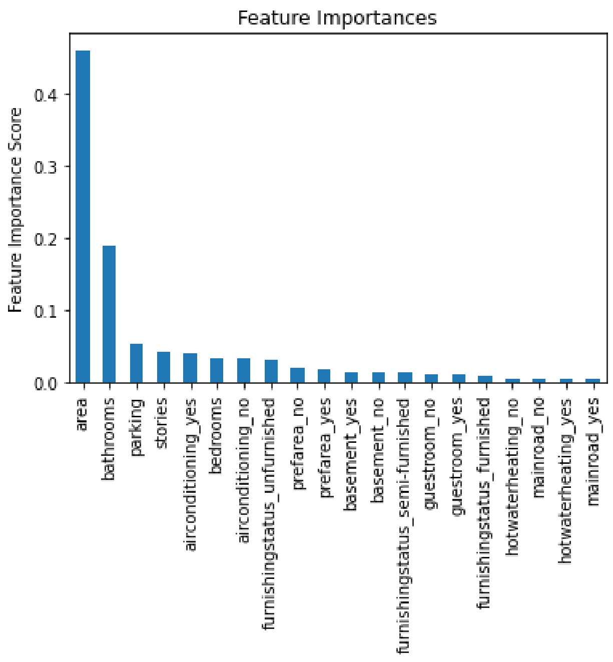
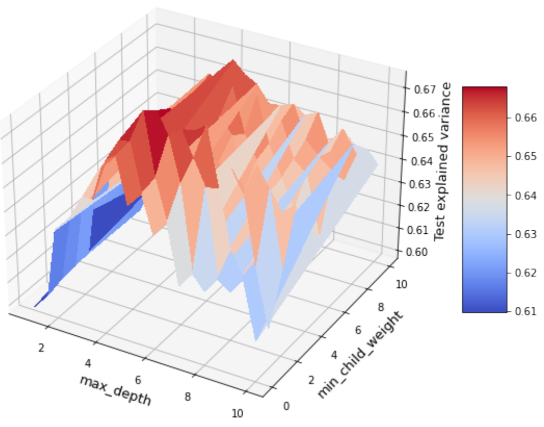


Figure 5: Variable importance for random forest regression model

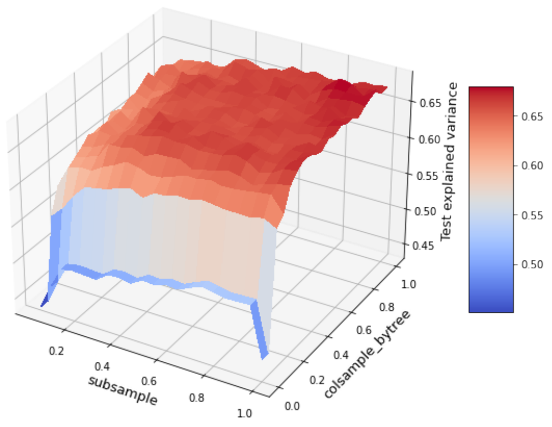
Boosting

After executing a 5 fold cross-validation to find the initial optimal number of estimators (81), the first grid search was executed. Figure 6(a) shows the results and the optimal values for `max_depth` and `min_child_weight` were used for further analysis. Subsequently, the number of estimators was re-calibrated again using a 5 fold cross-validation, and this resulted in 55 estimators. Next, `min_split_loss` was tuned, and this resulted in a value of 0. Subsequently, the number of estimators was re-calibrated and afterwards the tuning grid for `sample` and `colsample_bytree` was executed. Figure 6(b) shows the tuning results. Subsequently, using the same procedure grid search 4 and 5 were executed. Figure 6(d) shows the result of grid search 5. The 5 step sequential grid search with 5-fold cross-validation resulted in the following optimal parameters: `learning_rate` = 0.1, `n_estimators` = 100, `max_depth` = 4, `min_child_weight` = 3, `min_split_loss` = 0, `col_sample_bytree` = 0.9, `subsample` = 0.85, `req_alpha` = 100. The final model explained .580 variance on the test set. Figure 6(d) shows that the optimal learning rate should not be too small, neither too large. This makes perfect sense, since a small learning rate would not reach an optimal solution, but a large learning rate often reach a non-optimal local minimum.

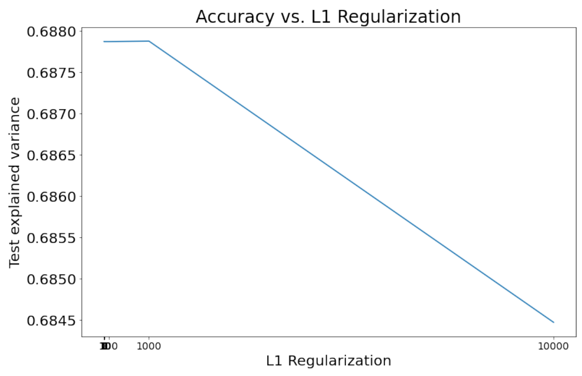
Figure 7 shows the variable importance, and again `area` is the most important variable, followed by `parking`, and `stories`. Interesting enough, these are not the same predictors which were found by the linear regression and random forest.



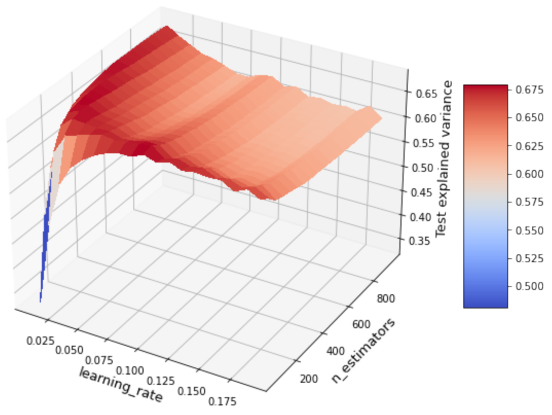
(a) `max_depth & min_child_weight`



(b) `subsample & colsample_bytree`



(c) `reg_alpha`



(d) `learning_rate & n_estimators`

Figure 6: Explained variance as function of a. max depth and minimum child weight, b. the subsample and colsample by tree, c. the L1 regularization term d. The learning rate and number of estimators.

Classification

Baseline: multinomial logistic lasso regression

Commonly used exploratory data analysis for high dimensional data includes histograms of average protein intensities for each participant and for each protein. For our breast cancer data set, these are presented in Figure 8.

We don't observe any abnormalities, so we can turn to multinomial logistic lasso regression. The model was fitted using `LogisticRegression`, with an L1-penalty, no penalization on the intercept, and a multinomial class. As measure of accuracy we use the proportion of correctly classified observations, which was .750 on the test set.

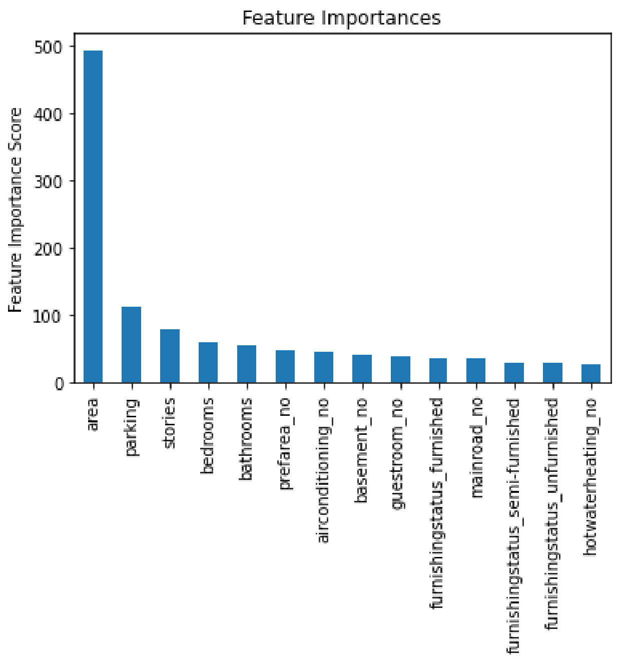


Figure 7: Variable importance for boosting model

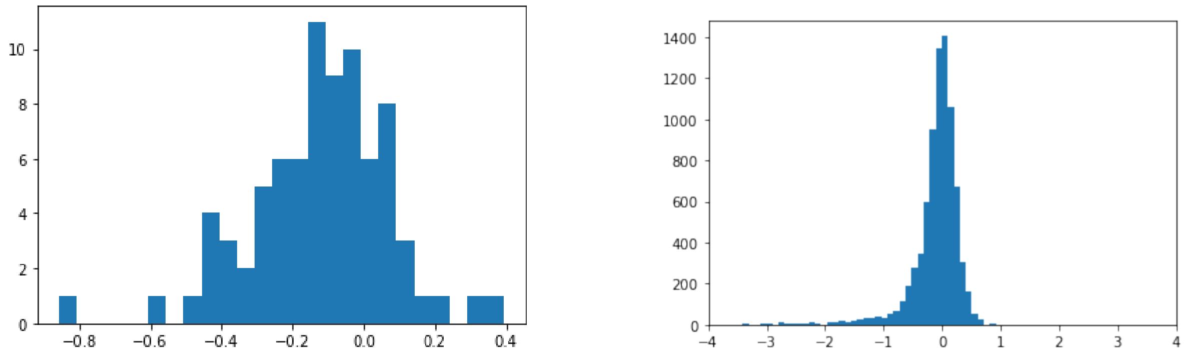
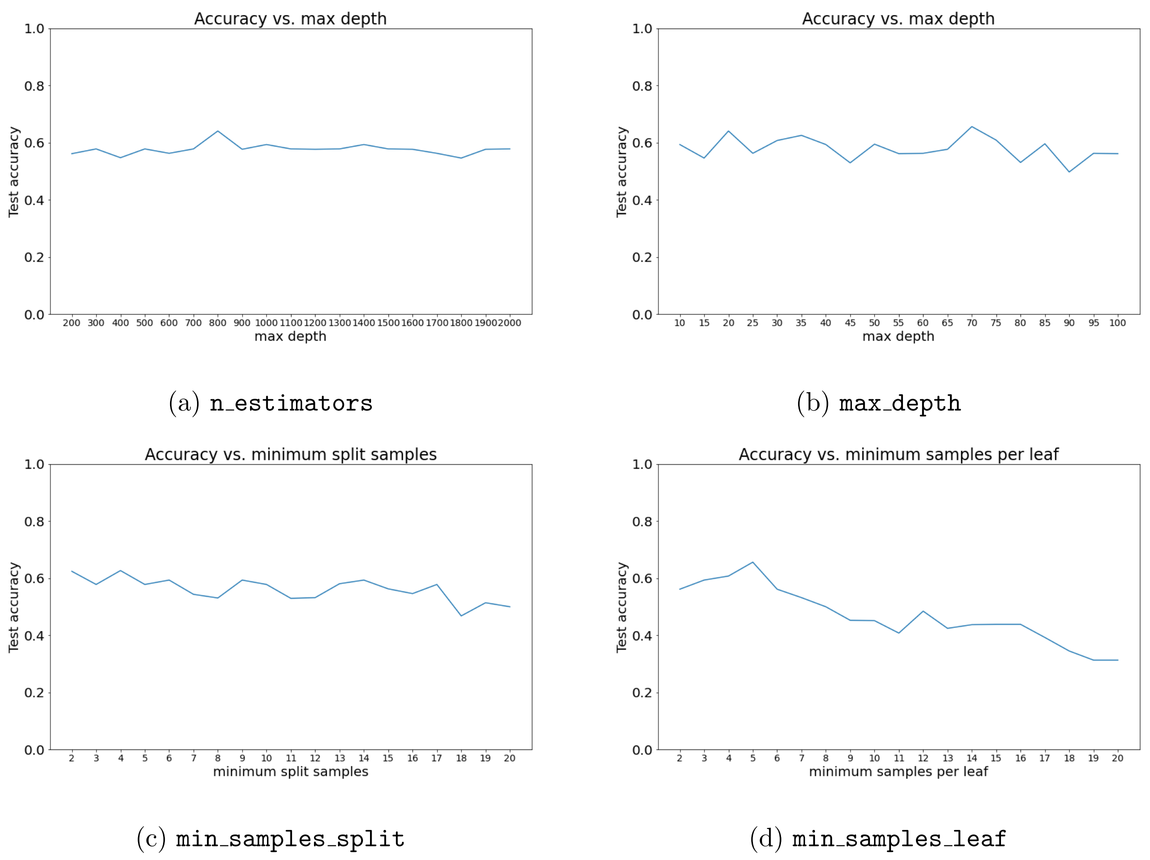


Figure 8: Average protein intensities per participant (left panel) and per protein (right panel)

Random forests

Random forests for classification was implemented in the same way as for regression, with a similar two step grid search procedure for tuning the hyperparameters. Instead of `RandomForestRegressor`, we now utilized `RandomForestClassifier`, with as final parameters `n_estimators` = 100, `max_depth` = 90, `min_samples_leaf` = 2 and `min_samples_split` = 3. Figure 9 shows how the classification accuracy changes in the parameter space for each hyper parameter. As with random forests for regression, we see that the the number of trees and the depth of the tree don't influence the final performance. The number of samples per split and the number of samples per leaf do influence the accuracy: if they increase, the accuracy decrease.

Using the final model to make predictions on the test set, yielded an accuracy of .875. Figure 10 present the thirty most important variables in classifying the different types of breast cancer.



(a) `n_estimators` (b) `max_depth`

(c) `min_samples_split` (d) `min_samples_leaf`

Figure 9: Classification accuracy as function of a. the number of trees, b. the maximum depth, c. the minimum number of samples per split and d. the minimum number of samples per leaf.

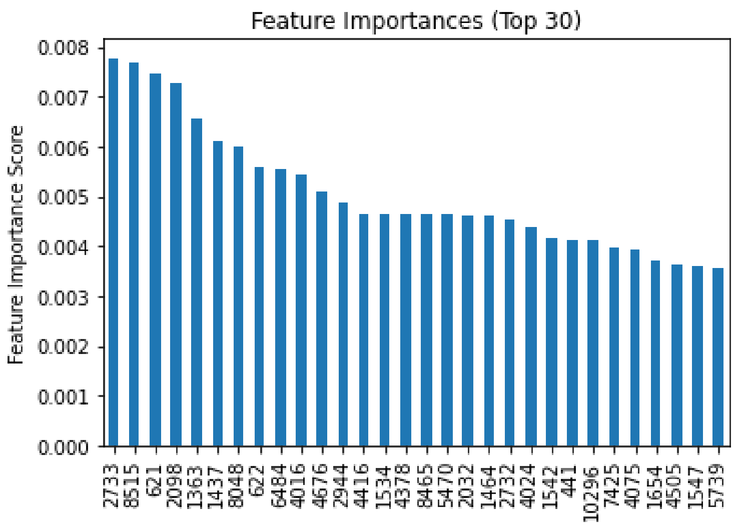
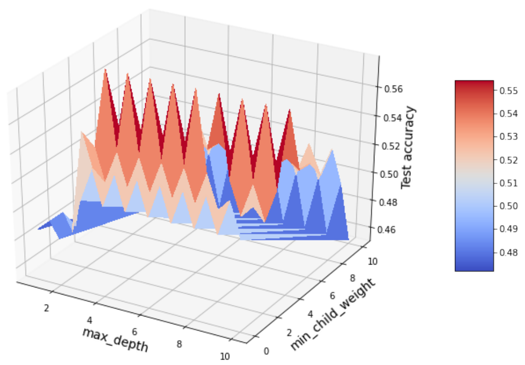


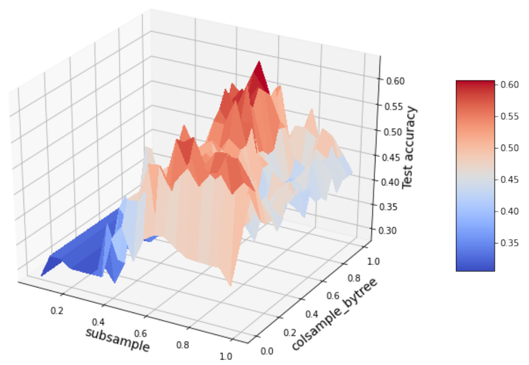
Figure 10: Top 30 most important proteins for classifying the four different types of breast cancer.

Boosting

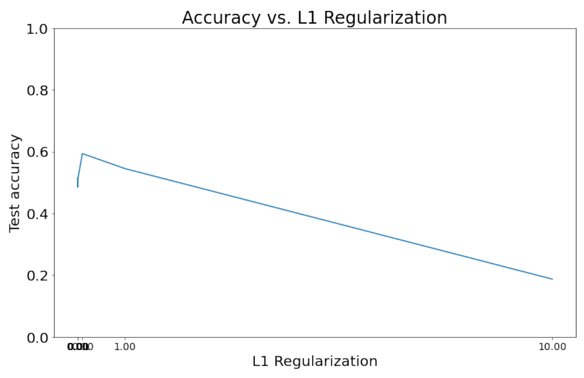
The first cross-validation showed that the intial optimal amount of estimators was 43. Subsequently, the max depth and min child weight parameters were tuned. Figure 11(a) shows the result. And again, after each grid search, the number of estimators was re-calibrated and all grid searches were executed sequential. Figure 11 shows the results of these grid searches. Figure 11(b) shows that parameter tuning is very important for boosting. Furthermore, it illustrates that it is important to use grids which combine parameters, since the performance is related to the interaction of both parameters. The 5 step sequential grid search with 5-fold cross-validation resulted in the following optimal parameters: `learning_rate` = 0.15, `n_estimators` = 500, `max_depth` = 4, `min_child_weight` = 3, `min_split_loss` = 0, `col_sample_bytree` = 0.8, `subsample` = 0.55, `req_alpha` = 0.1. The final model had a test accuracy of 0.875. Figure 12 shows that the variable 2732 is the most important predictor followed by 2032, and 3907.



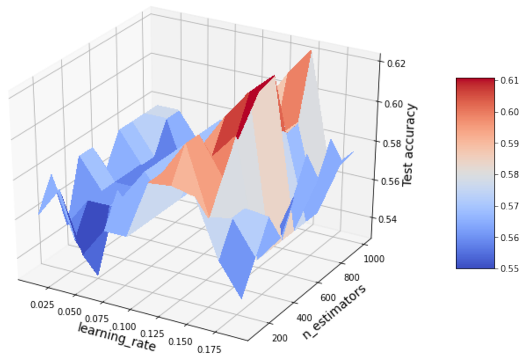
(a) `max_depth & min_child_weight`



(b) `subsample & colsample_bytree`



(c) `reg_alpha`



(d) `learning_rate & n_estimators`

Figure 11: Classification accuracy as function of a. max depth and minimum child weight, b. the subsample and colsample by tree, c. the L1 regularization term d. The learning rate and number of estimators.

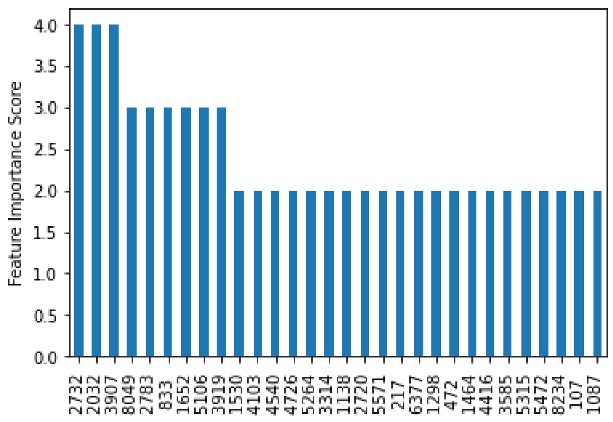


Figure 12: Top 30 most important proteins for classifying using for the boosting model

PCA

Figure 13 presents the results of Principal Components for our breast cancer data. Note that we first standardized our data by subtracting the mean and dividing by the standard deviation. PCA requires such a transformation, since it is a scale-dependent technique.

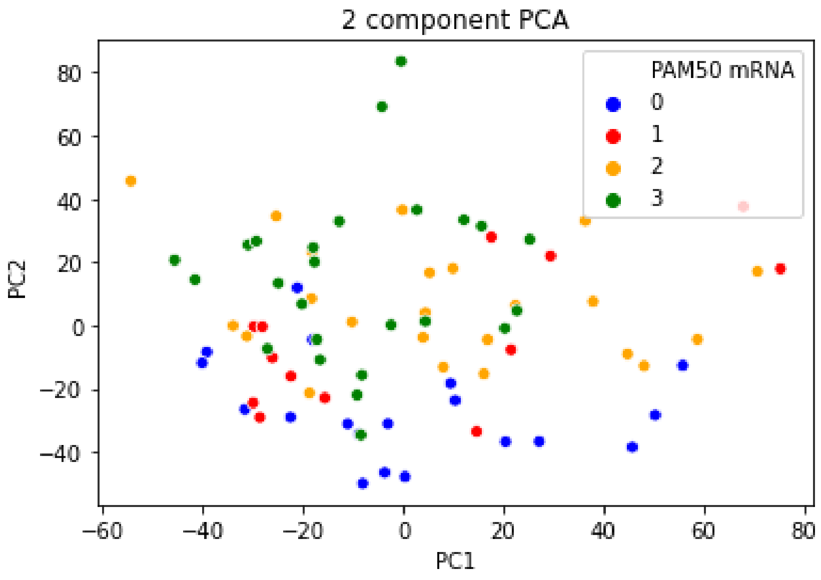
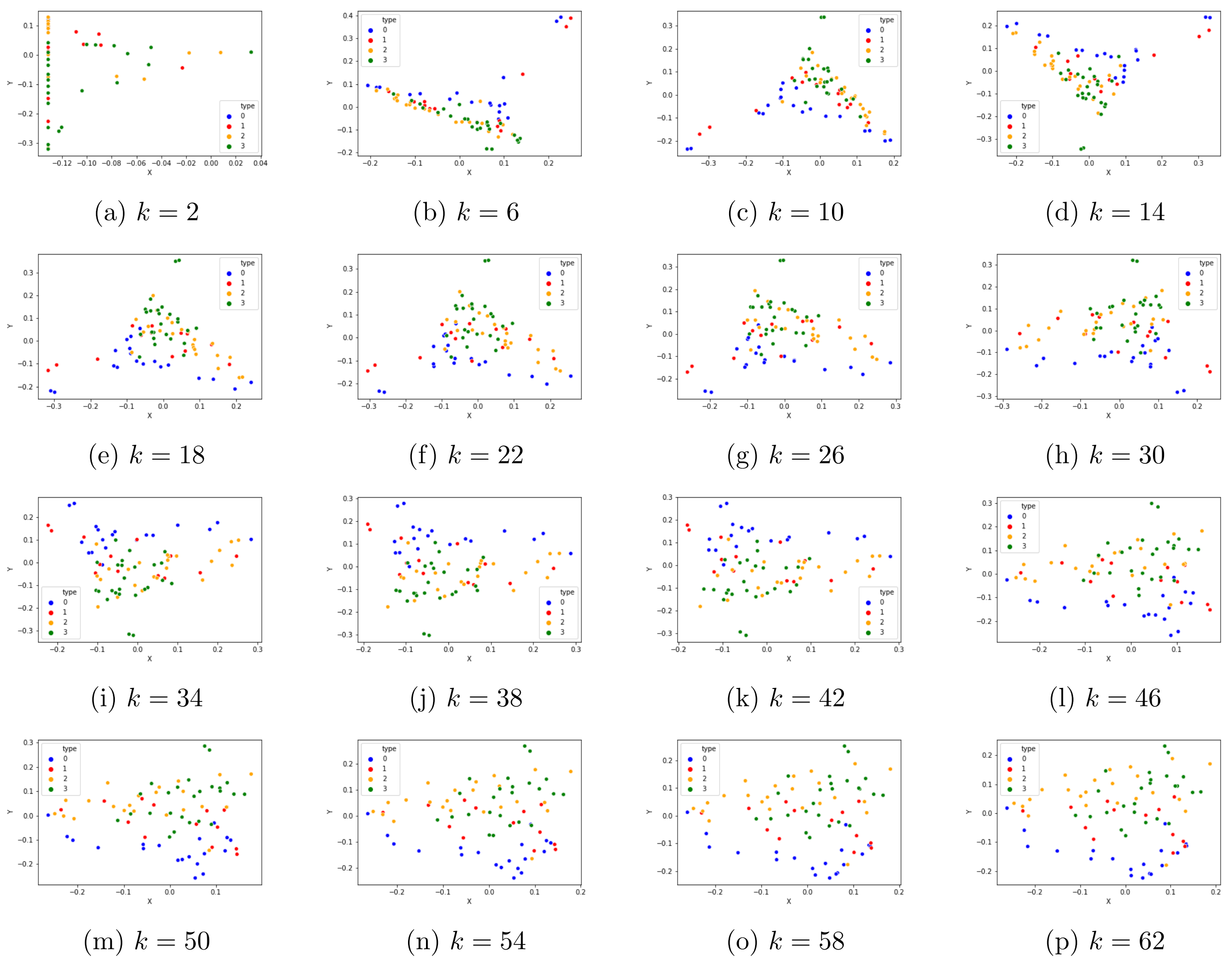


Figure 13: Two-dimensional Principal Component Analysis representation, with colors representing type of breast cancer (PAM50 mRNA)

Figure 13 shows that PCA does not work particularly well for this data set. We doe see that type 0 patients (Basal-like) have lower scores on the second dimension, that type 3 patients seem to produce higher scores in the second dimension and smaller scores on the first dimension and that both type 1 and type 2 patients score somewhere in the middle of the second dimension. However, there seems to be no clear distinctions. Let's see whether LLE does a better job!

LLE

Unlike PCA, LLE does have a hyper parameter to tune: the number of neighbors that are used to reconstruct points in a lower dimensional space, conveniently called k here. The effect of k on the representation of points can best be depicted graphically, and they are presented in Figure 14. As the number of neighbors increases, we see that the algorithm becomes better at clustering the groups. However, there seems to be some ceiling effect, because from $k = 14$ onwards, the clustering does not show much significant improvements. To end on a positive note: the clustering is actually quite better compared to PCA!



(a) $k = 2$ (b) $k = 6$ (c) $k = 10$ (d) $k = 14$

(e) $k = 18$ (f) $k = 22$ (g) $k = 26$ (h) $k = 30$

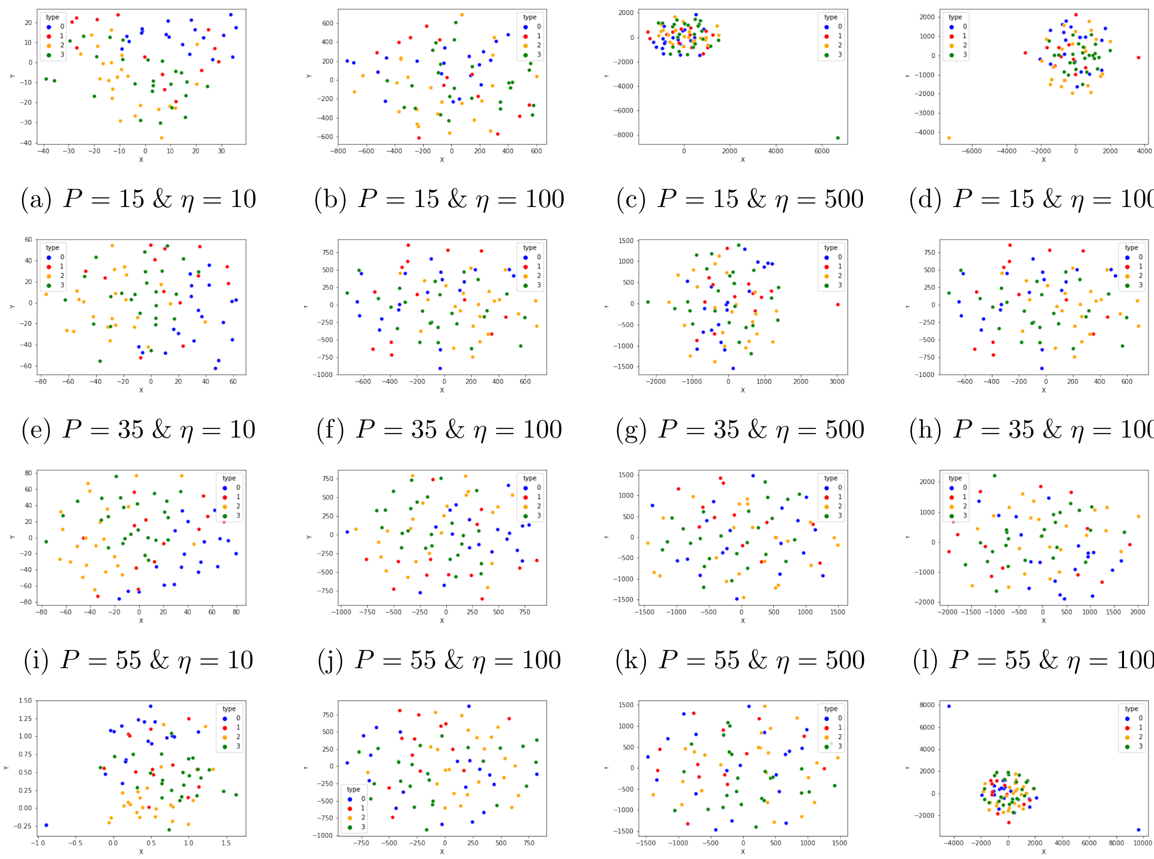
(i) $k = 34$ (j) $k = 38$ (k) $k = 42$ (l) $k = 46$

(m) $k = 50$ (n) $k = 54$ (o) $k = 58$ (p) $k = 62$

Figure 14: Two-dimensional LLE representations for different k.

t-SNE

T-SNE has multiple parameter that can be tuned. We decided to tune the perplexity and learning, while keeping the number of iterations constant at 10000, and after 1000 iterations without progression the algorithm stops automatically. Perplexity is usually between 5 and 50 and can not be higher than the number of observations. The learning rate is usually between 10 and 1000. Therefore we decided to select perplexity values of 15, 35, 55, and 75, and learning rates of 10, 100, 500, and 100. Figure 15 shows the resulting representations and we can conclude that $P = 55$ and $\eta = 10$ results in the best representations. Nearly all observations from class 0, 2, and 3 are separable. Nevertheless, the best t-SNE representation is not able to separate the data points from class 1. The clustering of the best t-SNE representation is slightly better than the best LLE representation.



(a) $P = 15$ & $\eta = 10$ (b) $P = 15$ & $\eta = 100$ (c) $P = 15$ & $\eta = 500$ (d) $P = 15$ & $\eta = 1000$

(e) $P = 35$ & $\eta = 10$ (f) $P = 35$ & $\eta = 100$ (g) $P = 35$ & $\eta = 500$ (h) $P = 35$ & $\eta = 1000$

(i) $P = 55$ & $\eta = 10$ (j) $P = 55$ & $\eta = 100$ (k) $P = 55$ & $\eta = 500$ (l) $P = 55$ & $\eta = 1000$

(m) $P = 75$ & $\eta = 10$ (n) $P = 75$ & $\eta = 100$ (o) $P = 75$ & $\eta = 500$ (p) $P = 75$ & $\eta = 1000$

Figure 15: Two-dimensional t-SNE representations for different perplexity P and learning rate η.

Conclusion

In this report we first presented an introduction to the different machine learning methods we have successfully implemented: random forests, boosting, PCA, LLE and t-SNE. We gave an elaborate description of the study design and we provided some background information on the different techniques. Subsequently we presented the results for regression as well as for classification, and found out that - for our data - both random forest and boosting did not outperform our baseline models of multiple linear regression and multinomial logistic lasso regression. For the visualization techniques we can conclude that t-SNE did best in representing the data in a two-dimensional space, followed up by LLE and PCA, respectively.

References

Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5–32.

Maaten, L. v. d., & Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, *9*(Nov), 2579–2605.

Roweis, S. T., & Saul, L. K. (2000). Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science*, *290*(5500), 2323-2326. doi: 10.1126/science.290.5500.2323

Saul, L. K., & Roweis, S. T. (2000). An introduction to locally linear embedding. *unpublished. Available at: http://www. cs. toronto. edu/˜ roweis/lle/publications. html*.